Pipeline and Cache for Processing Data Progressively

Field of Invention

[01] The invention relates generally to computer architectures, and more particularly to processing pipelines and caches.

Background

[02] As shown in Figure 1, processing pipelines are well known. A processing pipeline 100 includes stages 111-115 connected serially to each other. A first stage receives input 101, and a last stage 115 produces output 109. Generally, the output data of each stage is sent as input data to a next stage. The stages can concurrently process data. For example, as soon as one stage completes processing its data, the stage can begin processing next data received from the previous stage. As an advantage, pipelined processing increases throughput, since different portions of data can be processed in parallel.

[03] As shown in Figure 2, caches 200 are also well known. When multiple caches 211-215 are used, they are generally arranged in a hierarchy. The cache 215 'closest' to a processing unit 210 is usually the smallest in size and the fastest in access speed, while the cache 211 'farthest' from the processing unit is the largest and the slowest. For example, the cache 215 can be an 'on-chip' instruction cache, and the cache 211 a disk storage unit. As an advantage, most frequently used data are readily available to the processing unit.

[04] It is also known how to combine pipelines and caches.

[05] United States Patent 6,453,390, Aoki, et al., September 17, 2002, "Processor cycle time independent pipeline cache and method for pipelining data from a cache," describes a processor cycle time independent pipeline cache and a method for pipelining data from a cache to provide a processor with operand data and instructions without introducing additional latency for synchronization when processor frequency is lowered or when a reload port provides a value a cycle earlier than a read access from the cache storage. The cache incorporates a persistent data bus that synchronizes the stored data access with the pipeline. The cache can also utilize bypass mode data available from a cache input from the lower level when data is being written to the cache.

[06] United States Patent 6,427,189, Mulla, et al., July 30, 2002, "Multiple issue algorithm with over subscription avoidance feature to get high bandwidth through cache pipeline," describes a multi-level cache structure and associated method of operating the cache structure. The cache structure uses a queue for holding address information for memory access requests as entries. The queue includes issuing logic for determining which entries should be issued. The issuing logic further includes first logic for determining which entries meet a predetermined criteria and selecting a plurality of those entries as issuing entries. The issuing logic also includes last logic that delays the issuing of a selected entry for a predetermined time period based upon a delay criteria.

[07] United States Patent 5,717,896, Yung, et al., February 10, 1998, "Method and apparatus for performing pipeline store instructions using a single cache access pipestage," describes a mechanism for implementing a store instruction so that a single cache access stage is required. Since a load instruction requires a single cache access stage, in which a cache read occurs, both the store and load instructions utilize a uniform number of cache access stages. The store instruction is implemented in a pipeline microprocessor such that during the pipeline stages of a given store instruction, the cache memory is read and there is an immediate determination if there is a tag hit for the store. Assuming there is a cache hit, the cache write associated with the given store instruction is implemented during the same pipeline stage as the cache access stage of a subsequent instruction that does not write to the cache or if there is no instruction. For example, a cache data write occurs for the given store simultaneously with the cache tag read of a subsequent store instruction.

[08] United States Patent 5,875,468, Erlichson, et al., February 23, 1999, "Method to pipeline write misses in shared cache multiprocessor systems," describes a computer system with a number of nodes. Each node has a number of processors which share a single cache. A method provides a release consistent memory coherency. Initially, a write stream is divided into separate intervals or epochs at each cache, delineated by processor synch operations. When a write miss is detected, a counter corresponding to the current epoch is incremented. When the write miss globally completes, the same epoch counter is decremented. Synch operations issued to the cache stall the issuing processor until all epochs up to and including the epoch that

the synch ended have no misses outstanding. Write cache misses complete from the standpoint of the cache when ownership and data are present.

[09] United States Patent 5,283,890, Petolino, Jr., et al., February 1, 1994, "Cache memory arrangement with write buffer pipeline providing for concurrent cache determinations," describes a cache memory that is arranged using write buffering circuitry. This cache memory arrangement includes a Random Access Memory (RAM) array for memory storage operated under the control of a control circuit which receives input signals representing address information, write control signals, and write cancel signals.

Summary of Invention

[010] A system for processing data includes a processing pipeline, a progressive cache, and a cache manager.

[011] The processing pipeline includes stages connected serially to each other so that an output element of a previous stage is sent as an input element to a next stage.

[012] A first stage is configured to receive a processing request for input. A last stage is configured to produce output corresponding to the input.

[013] The progressive cache includes caches arranged in an order from least finished cache elements to most finished cache elements. Each cache of the progressive cache receives an output cache element of a corresponding stage

of the processing pipeline and sends an input cache element to a next stage after the corresponding stage.

[014] The cache controller routes cache elements from the processing pipeline to the progressive cache in the order from a least finished cache element to a most finished cache element and from the progressive cache to the processing pipeline in the order from the most finished cache element to the next stage after the corresponding stage.

Brief Description of the Drawings

[015] Figure 1 is a block diagram of a prior art processing pipeline;

[016] Figure 2 is a block diagram of a prior art hierarchical cache; and

[017] Figure 3 is a block diagram of a pipeline with a progressive cache according to the invention.

Detailed Description of a Preferred Embodiment of the Invention

[018] System Structure

[019] Figure 3 shows a system 300 for efficiently processing data. The system 300 includes a processing pipeline 310, a cache manager 320, and a progressive cache 330.

[020] The pipeline 310 includes processing stages 311-315 connected serially to each other. The first stage 311 receives input 302 for a processing request 301. The last stage 315 produces output 309. Each stage can provide output for the next stage, as well as to the cache manager 320.

[021] The cache manager 320 connects the pipeline 310 to the progressive cache 330. The cache manager routes cache elements between the pipeline and the progressive cache.

[022] The progressive cache 330 includes caches 331-335. There is one cache for each corresponding stage of the pipeline. The progressive caches 331-335 are arranged, left-to-right in the Figure 3, from a least finished, i.e., least complete, cache element to a most finished, i.e., most complete, cache element, hence, the cache 330 is deemed to be 'progressive'. Each cache 331-335 includes data for input to a next stage of a corresponding stage in the pipeline 310 and for output from the corresponding stage.

[023] The one-to-one correspondences between the processing stages of the pipeline and the caches of the progressive cache are indicated generally by the dashed double arrows 341-345.

[024] The stages increase a level of completion of elements passing through the pipeline, and there is a cache for each level of completion. For the purpose of this description, the caches are labeled types 1-5.

[025] System Operation

[026] First, the processing request 301 for the input 302 is received.

[027] Second, the progressive cache 330 is queried 321 by the cache manager 320 to determine a most complete cached element representing the output 309, e.g., cached elements contained in caches 351-355 of cache types 1-5, which is available to satisfy the processing request 301.

[028] Third, a result of querying the progressive cache 330, i.e., the most complete cached element, is sent, i.e., piped, to the appropriate processing stage, i.e., the next stage of the corresponding stage of the pipeline 310, to complete the processing of the data. This means that processing stages can be by-passed. If no cache element is available, then processing of the processing request commences in stage 311. If the most completed element corresponds to the last stage, then no processing needs to be done at all.

[029] After each stage completes processing, the output of the stage can also be sent, i.e., piped, back to the progressive cache 330, via the cache manger 320, for potential caching and later reuse.

[030] As caches fill, least recently used (LRU) cache elements can be discarded. Cache elements can be accessed by hashing techniques.

[031] In another embodiment of the system 300, there are fewer caches in the progressive cache 330 than there are stages in the processing pipeline 310. In this embodiment, not all stages have a corresponding cache. It is

sometimes advantageous to eliminate an individual cache in the progressive cache 330 because the corresponding stage is extremely efficient and caching the output in the individual cache would be unnecessary and would waste memory. Furthermore, the output of the corresponding stage may require too much memory to be practical.

[032] One skilled in the art would readily understand how to adapt the system 300 to include various processing pipelines and various progressive caches to enable a processing request to be satisfied.

[033] Although the invention has been described by way of examples of preferred embodiments, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.